

# Fast matrix decomposition in $\mathbb{F}_2$

Enrico Bertolazzi

Department of Mechanical and Structural Engineering – University of Trento, Italy,

Anna Rimoldi

Department of Mathematics – University of Trento, Italy.

---

In this work an efficient algorithm to perform a block decomposition (and so to compute the rank) of large dense rectangular matrices with entries in  $\mathbb{F}_2$  is presented. Depending on the way the matrix is stored, the operations acting on rows or block of consecutive columns (stored as one integer) should be preferred. In this paper, an algorithm that completely avoids the column permutations is given. In particular, a block decomposition is presented and its running times are compared with the ones adopted into SAGE [Stein et al. 2012].

Categories and Subject Descriptors: [Numerical Analysis]: Matrix decomposition

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Software implementation of finite field arithmetic, mathematical tools for cryptography, Linear Algebra, rank computation, matrix decomposition.

---

## 1. INTRODUCTION

An important tool in linear algebra is the matrix decomposition, which expresses a (rectangular) matrix as a product of two or more simpler matrices. Such decompositions are used for easy computation of rank, null space, and solving linear system and related problem.

There are many well-known algorithms for decomposition defined in any field, finite or not. A common approach consists in reducing a matrix usually to the row-echelon form by row operations [Meyer 2000]. Once the row-echelon form is obtained the rank will be equal to the number of non-zero rows and null space can be easily computed. Gauss *LU* decomposition [Golub and Van Loan 1996] can be also used to solve linear systems (when the matrix is square and full rank) or compute the rank. Applied to rectangular or rank deficient matrices, it is costly as the computation of the row-echelon form.

In fact, Gauss decomposition of a matrix  $A$  produces an *LU* factorization, i.e.  $PAQ = LU$  where  $P$  and  $Q$  are permutation matrices, and  $U$  is the row-echelon form of  $A$  (up to column permutation). The asymptotic cost of a naive implementation of *LU* decomposition for a dense  $n \times n$  matrix is  $\mathcal{O}(n^3)$ . However such a cost can be reduced using a combination of recursion and matrix-matrix multiplication.

For example, using matrix-matrix multiplication (MMM) in the construction of the factorization, the asymptotic cost may be reduced by using faster MMM algorithms. The problem of fast matrix-matrix multiplication is still an open problem.

The naive MMM algorithm is based on the classical definition of the multiplication of two matrices; its cost is  $n^3$  multiplications and  $n^2(n - 1)$  additions and so we classify the naive algorithm as an  $\mathcal{O}(n^3)$  algorithm.

Strassen matrix-matrix multiplication algorithm [Strassen 1969] – which asymptotic cost is  $\mathcal{O}(n^{\log_2 7})$  in any field – uses only seven scalar multiplications (instead of the usual eight) to multiply  $2 \times 2$  matrices. In fact, as proved in [Winograd 1971], Strassen’s algorithm is optimal for  $2 \times 2$  matrices. Further asymptotic improvements [Coppersmith and Winograd 1990] can be obtained to perform multiplication of larger matrices. Hybrid algorithms incorporate Strassen and Winograd variants recursively to achieve high performance on large matrices [Huss-Lederman et al. 1996; Higham 1990; Douglas et al. 1994; Kaporin 1999]. The asymptotic cost of  $\mathcal{O}(n^{\log_2 7})$  means that for a large enough  $n$ , Strassen’s algorithm should theoretically perform multiplication significantly faster than the naive algorithm. However, asymptotic cost means that the actual cost of standard  $LU$  decomposition is about  $C_1 n^3$  while the actual cost of Strassen multiplication is about  $C_2 n^{\log_2 7}$  where  $C_2 \gg C_1$ . Therefore, the use of Strassen algorithm is convenient only for large  $n$ . Strassen algorithm is recursive so that normally the recursion is terminated when the cost of recursion is larger than the classical matrix-matrix multiplication. That happens when  $C_2 n^{\log_2 7} \approx C_1 n^3$ , i.e.  $n \approx \exp(\ln(C_2/C_1)/(3 - \log_2 7))$ . For instance, if  $C_2/C_1 \approx 10$  we have  $n \approx 150000$  while in case  $C_2/C_1 \approx 5$  we have  $n \approx 4000$ . In practice, the computation of the switching point must consider additional costs and it is implementation dependent. For a detailed analysis see for example [Huss-Lederman et al. 1996; Higham 1990].

The efficient computation of MMM can be further improved in case of finite field and in particular for finite field  $\mathbb{F}_2$  the *Method of four Russian for Multiplication* (M4RM) is a fast MMM algorithm, which cost is  $\mathcal{O}(n^3 / \log n)$  [Arlazarov et al. 1970; Aho et al. 1974; Albrecht et al. 2010]. Its asymptotic cost is better than classical matrix-matrix multiplication; but it is worse than recursive Strassen’s algorithm. However, if the actual cost of M4RM is about  $C_3 n^3 / \log n$ , we have that  $C_3 \ll C_1$  so that is competitive for not too big matrices. A combination of Strassen and M4RM is a good compromise for faster matrix-matrix multiplication [Albrecht et al. 2010].

The fast decomposition of an  $n \times m$  matrix with entries in a finite field  $\mathbb{F}_2$  is an important issue in algorithmic number theory and cryptanalysis [Shoup 2009; Bach and Shallit 1996]. In fact, some problems in cryptanalysis and number theory can be transformed in one involving a linear system with entries in  $\mathbb{F}_2$ . The existence of solutions of a linear system can be deduced by analyzing the rank of the corresponding matrix.

In this paper we propose a new efficient algorithm to perform the matrix factorization for large dense rectangular matrices with entries in  $\mathbb{F}_2$ . It uses an efficient implementation of M4RM algorithm and uses only row permutations. In section 2 our notation is given and an appropriate data structure to store and manipulate the matrix is described. In section 3 the non-recursive block algorithm is presented. In section 4 the recursive version of the main algorithm is described. In section 5 some details about the choice of some parameters are given. In section 6 some tests comparing our algorithm with Sage packages [Stein et al. 2012] are presented.

## 2. THE USED MATRIX DATA STRUCTURE

Let  $\mathbf{A}$  be an  $n \times m$  matrix having entries in  $\mathbb{F}_2$ , i.e.  $\mathbf{A} \in \text{Mat}(n, m, \mathbb{F}_2)$ , denoted as

$$\mathbf{A} = \begin{pmatrix} a_{0,0} & \cdots & a_{0,m-1} \\ \vdots & \ddots & \vdots \\ a_{n-1,0} & \cdots & a_{n-1,m-1} \end{pmatrix}.$$

We adopt the following convention for intervals of indices  $a..b = \{a, a+1, \dots, b-1\}$  and so, for instance, the sub-matrix  $\mathbf{A}_{a..b, c..d} \in \text{Mat}(b-a, d-c, \mathbb{F}_2)$  of  $\mathbf{A}$  represents the intersection of rows of index from  $a$  to  $b-1$  and columns of index from  $c$  to  $d-1$ . The sub-matrix  $\mathbf{A}_{a..b, \bullet} \in \text{Mat}(b-a, m, \mathbb{F}_2)$  of  $\mathbf{A}$  is composed by the rows of index from  $a$  to  $b-1$ . Furthermore, we denote by  $\text{rank}(\mathbf{A})$  the rank of  $\mathbf{A}$ , i.e. the maximum number of linearly independent row (or column) vectors of  $\mathbf{A}$ .

We are interested in the computation of the factorization of large dense matrices that do not fit into the cache. Thus, a good arrangement of the elements of the matrix in memory is important for an efficient data retrieval. Moreover, bits are naturally grouped in words whose size is a power of 2, typically 32, 64 or 128 for larger architectures. An element of  $\mathbb{F}_2$  is naturally represented as one bit, so that elements in  $\mathbb{F}_2$  are naturally grouped in words of 32, 64 or 128 bits. In particular, a string of elements in  $\mathbb{F}_2$  is packed in an (unsigned) integer. The advantage of storing multiple elements of  $\mathbb{F}_2$  as an integer is that it guarantees a natural parallelism of some operations.

From now on, we denote with  $b$  the number of bits of the computer architecture, i.e. the number of bits in one machine word. Given two integers  $x$  and  $y$  whose bits represent elements in  $\mathbb{F}_2$ , the operation of *exclusive or*, denoted by  $x \oplus y$ , is the sum  $\oplus$  in  $\mathbb{F}_2$  applied to  $x$  and  $y$  bitwise; the *and* operation, denoted by  $x \odot y$ , is the multiplications  $\odot$  in  $\mathbb{F}_2$  applied to  $x$  and  $y$  bitwise. Infact, we have the following formulas

$$x = \sum_{i=0}^b x_i 2^i, \quad y = \sum_{i=0}^b y_i 2^i, \quad x \oplus y = \sum_{i=0}^b (x_i \oplus y_i) 2^i, \quad x \odot y = \sum_{i=0}^b (x_i \odot y_i) 2^i.$$

The entries of a matrix are packed into integers that represent groups of elements of the matrix itself. In particular,  $b$  consecutive entries in one row are packed into one integer. The way the integers are arranged changes how you access the elements of the matrix and you implement the elementary operations.

A matrix  $\mathbf{A} \in \text{Mat}(n, m, \mathbb{F}_2)$  is stored into a matrix of non-negative integers  $\mathcal{A} \in \text{Mat}(n, \mu, \mathbb{N})$  with  $\mu b - b < m \leq \mu b$ . To access to the element  $a_{ij}$  of  $\mathbf{A}$ , we have to determine the corresponding column-block  $q$  and then the right bit. Precisely, using integer division with remainder  $j = qb + r$ , ( $0 \leq r < b$ ), the element  $a_{ij}$  corresponds to the  $r$ -th bit of the integer  $\mathcal{A}_{iq}$ .

*Remark 2.1.* According to the previous notation, the least significant bit is on the left respect to the most significant bit, i.e. we are using big endian bit order.

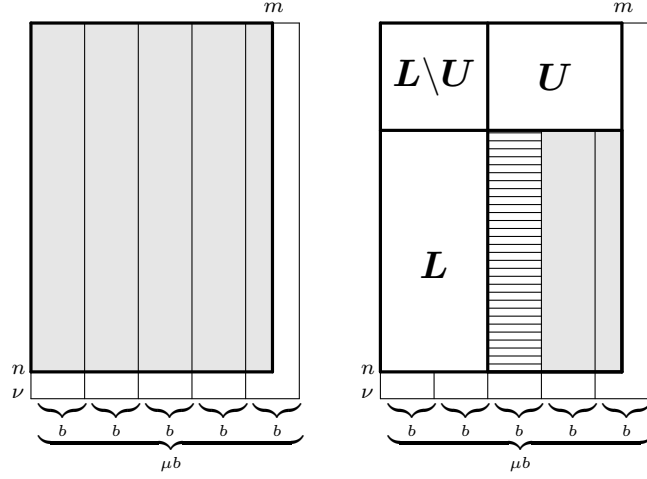


Fig. 1.

We consider a trivial example. Let  $b = 3$  and  $\mathbf{A}$  be the following  $(4 \times 5)$ -matrix:

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Since  $m = 5$  is not a multiple of  $b = 3$ , we directly memorize the matrix

$$\mathbf{A}' = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

in a  $4 \times 2$  matrix of 3 bit integer as follows

$$\mathcal{A} = \begin{pmatrix} 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 & 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 \\ 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 & 0 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 \\ 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 & 1 \cdot 2^0 + 0 \cdot 2^1 + 0 \cdot 2^2 \\ 0 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2 & 1 \cdot 2^0 + 1 \cdot 2^1 + 0 \cdot 2^2 \end{pmatrix} = \begin{pmatrix} 5 & 3 \\ 1 & 2 \\ 3 & 1 \\ 4 & 3 \end{pmatrix}. \quad (1)$$

The elements of  $\mathcal{A}$  can be organized using row-major order or column-major order. Our algorithm works better using column-major order. For example matrix  $\mathcal{A}$  in (1) is stored as

$$(5 \ 1 \ 3 \ 4 \mid 3 \ 2 \ 1 \ 3).$$

Due to the way the matrix is stored, it is clear that the operations acting on rows or block of  $b$  consecutive columns (stored as one integer) should be preferred. Operations acting on isolated columns should be avoided. In fact, the cost of an operation

on a single column on matrix  $A$  is about equal (or more) the cost of the same operation performed on a group of  $b$  columns when the columns are stored in a single column of the integer matrix  $A$ .

### 3. BLOCK DECOMPOSITION

In this section we give a (non-recursive) block decomposition that holds for matrices with entries in any field. For simplicity of notation, we are going to describe it for matrices in  $\mathbb{F}_2$ . Let consider  $A \in \text{Mat}(n, m, \mathbb{F}_2)$  such that it can be split in four blocks

$$A = \left[ \begin{array}{c|c} B & C \\ \hline D & E \end{array} \right],$$

where the block  $B \in \text{Mat}(r, b, \mathbb{F}_2)$ , with  $r \leq b$ , has full rank and the rows of  $D$  are linearly dependent on those of  $B$ . In other words, it holds that  $D = YB$ .

The factorization is based on the identity:

$$\left[ \begin{array}{c|c} B & C \\ \hline D & E \end{array} \right] = \left[ \begin{array}{c|c} I & 0 \\ \hline Y & I \end{array} \right] \left[ \begin{array}{c|c} I & 0 \\ \hline 0 & E \oplus YC \end{array} \right] \left[ \begin{array}{c|c} B & C \\ \hline 0 & I \end{array} \right]. \quad (2)$$

Due to the previous identity, we have that

$$\left[ \begin{array}{c|c} I & 0 \\ \hline 0 & E \oplus YC \end{array} \right] \left[ \begin{array}{c|c} B & C \\ \hline 0 & I \end{array} \right] = \left[ \begin{array}{c|c} B & C \\ \hline 0 & E \oplus YC \end{array} \right]$$

and notice that  $\text{rank}(A) = \text{rank}(E \oplus YC) + \text{rank}(B) = \text{rank}(E \oplus YC) + r$ . The sub-matrix  $E \oplus YC \in \text{Mat}(n - r, m - b, \mathbb{F}_2)$  is the Schur complement of  $A$  (see [Haynsworth 1968; Zhang 2005]). Thus, we have reduced the decomposition to a smaller problem. Applying the same idea as before we can reduce the decomposition to smaller and smaller problems with a reduction steps that can be described as follows.

Let  $A^{(0)} = A$  and  $P^{(0)}$  be a permutation matrix such that  $P^{(0)}A^{(0)}$  can be partitioned as

$$P^{(0)}A^{(0)} = \left[ \begin{array}{c|c} B^{(0)} & C^{(0)} \\ \hline D^{(0)} & E^{(0)} \end{array} \right], \quad \text{where } B^{(0)} \in \text{Mat}(r_0, b, \mathbb{F}_2), \quad \text{rank}(B^{(0)}) = r_0 \leq b$$

and the rows of  $D^{(0)}$  are linearly dependent on those of  $B^{(0)}$ , i.e.  $D^{(0)} = Y^{(0)}B^{(0)}$ . Using the products in (2) we obtain the following decomposition

$$P^{(0)}A^{(0)} = \left[ \begin{array}{c|c} I & 0 \\ \hline Y^{(0)} & I \end{array} \right] \left[ \begin{array}{c|c} I & 0 \\ \hline 0 & A^{(1)} \end{array} \right] \left[ \begin{array}{c|c} B^{(0)} & C^{(0)} \\ \hline 0 & I \end{array} \right],$$

where  $A^{(1)} = E^{(0)} \oplus Y^{(0)}C^{(0)} \in \text{Mat}(n - r_0, m - b, \mathbb{F}_2)$ . Now, we can apply the same idea to the Schur complement  $A^{(1)}$ . Let  $P^{(1)}$  be a permutation matrix such

that

$$P^{(1)}A^{(1)} = \left[ \begin{array}{c|c} B^{(1)} & C^{(1)} \\ \hline D^{(1)} & E^{(1)} \end{array} \right], \quad \text{where } B^{(1)} \in \text{Mat}(r_1, b, \mathbb{F}_2), \quad \text{rank}(B^{(1)}) = r_1 \leq b$$

and  $D^{(1)} = Y^{(1)}B^{(1)}$ . Due to the decomposition (2), we reduce further the problem of the decomposition of  $A^{(1)}$  to the decomposition of  $A^{(2)} = E^{(1)} \oplus Y^{(1)}C^{(1)} \in \text{Mat}(n - r_0 - r_1, m - 2b, \mathbb{F}_2)$ .

Going forward, at the  $j^{\text{th}}$  stage, the original matrix  $A$  is transformed into the sub-matrix  $A^{(j)}$ ; notice that  $\text{rank}(A) = \text{rank}(A^{(j)}) + r_0 + r_1 + \dots + r_{j-1}$ . Next, we find a permutation  $P^{(j)}$  such that  $P^{(j)}A^{(j)}$  can be partitioned as follows

$$P^{(j)}A^{(j)} = \left[ \begin{array}{c|c} B^{(j)} & C^{(j)} \\ \hline D^{(j)} & E^{(j)} \end{array} \right], \quad \text{with } B^{(j)} \in \text{Mat}(r_j, b, \mathbb{F}_2), \quad (3)$$

with  $r_j \leq b$  and  $B^{(j)}$  full rank and  $D^{(j)} = Y^{(j)}B^{(j)}$ . Using (2) again, we have

$$\left[ \begin{array}{c|c} I & 0 \\ \hline 0 & P^{(j)}A^{(j)} \end{array} \right] = \left[ \begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & I & 0 \\ \hline 0 & Y^{(j)} & I \end{array} \right] \left[ \begin{array}{c|c} I & 0 \\ \hline 0 & I \end{array} \right] \left[ \begin{array}{c|c|c} I & 0 & 0 \\ \hline 0 & B^{(j)} & C^{(j)} \\ \hline 0 & 0 & I \end{array} \right],$$

where

$$A^{(j+1)} = E^{(j)} \oplus Y^{(j)}C^{(j)}. \quad (4)$$

Note that  $A^{(j+1)}$  is the Schur complement of  $P^{(j)}A^{(j)}$  in  $\mathbb{F}_2$  and, after  $\mu$  steps, we obtain

$$PA = \underbrace{\left[ \begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & I \end{array} \right]}_L \left[ \begin{array}{c|c} I & 0 \\ \hline 0 & A^{(\mu-1)} \end{array} \right] \underbrace{\left[ \begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & I \end{array} \right]}_U, \quad (5)$$

where  $L$  is non-singular lower triangular matrix and  $U$  is the full rank upper block staircase. Let  $P^{(\mu-1)}$  be a permutation matrix such that  $P^{(\mu-1)}A^{(\mu-1)}$  can be partitioned as

$$P^{(\mu-1)}A^{(\mu-1)} = \left[ \begin{array}{c|c} B^{(\mu-1)} \\ \hline D^{(\mu-1)} \end{array} \right], \quad \text{where } B^{(\mu-1)} \in \text{Mat}(r_{\mu-1}, b', \mathbb{F}_2),$$

with  $b' = m - (\mu - 1)b$  that satisfy  $r_{\mu-1} \leq b' \leq b$  and  $B^{(\mu-1)}$  is full rank.

Moreover,  $D^{(\mu-1)} = Y^{(\mu-1)} B^{(\mu-1)}$  and the decomposition (5) becomes

$$\begin{aligned} \left[ \begin{array}{c|c} I & 0 \\ \hline 0 & P^{(\mu-1)} \end{array} \right] PA &= \left[ \begin{array}{c|c} L_{11} & 0 \\ \hline P^{(\mu-1)} L_{21} & I \end{array} \right] \left[ \begin{array}{c|c} I & 0 \\ \hline 0 & \left[ \begin{array}{c|c} I & \\ \hline Y^{(\mu-1)} & \end{array} \right] B^{(\mu-1)} \end{array} \right] \left[ \begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & I \end{array} \right], \\ &= \underbrace{\left[ \begin{array}{c|c} L_{11} & 0 \\ \hline P^{(\mu-1)} L_{21} & \left[ \begin{array}{c|c} I & \\ \hline Y^{(\mu-1)} & \end{array} \right] \end{array} \right]}_L \underbrace{\left[ \begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & B^{(\mu-1)} \end{array} \right]}_U, \end{aligned}$$

where  $L$  is full rank lower trapezoidal matrix and  $U$  is the full rank upper block staircase. The previous steps can be resumed in the Lemma:

LEMMA 3.1. *Given any (rectangular) matrix  $A$ , there exist a permutation  $P$  such that*

$$PA = LU$$

where  $U$  is full rank upper (block) triangular matrix and  $L$  is a full rank lower trapezoidal matrix.

Clearly,  $\text{rank}(A)$  is the rank of the matrix  $U$  which is the number of its rows:  $\text{rank}(A) = \sum_{j=0}^{\mu-1} r_j$ .

Observe that the computation at the step  $j$  involves matrix-matrix multiplications to obtain the Schur complement (4) which are the mostly costly operations of the presented algorithm. The multiplication of a  $p \times b$ -matrix by a  $b \times q$ -matrix, in case  $p \gg b$ , can be efficiently performed using the M4RM algorithm [Arlazarov et al. 1970; Aho et al. 1974; Albrecht et al. 2010]. Thus, in the computation of  $A^{(j+1)}$  it is convenient to use the M4RM algorithm, because this block operation is more efficient than the usual row operations.

This decomposition is based on the selection of  $B^{(j)}$  and the construction of  $Y^{(j)}$  at each step. Efficient algorithms for this will be discussed in the next sections. For this purpose the incremental construction of an inverse of a matrix and pseudo-inverse construction is a necessary tool.

*Remark 3.2.* The decomposition described in Lemma 3.1 when  $b = 1$  is equivalent to the PLE factorization described in [Albrecht et al. 2011; Jeannerod et al. 2011]. However computation with  $b = 1$  is not convenient losing natural parallelism of integer operations.

### 3.1 The computation of $Y^{(j)}$

The permutation  $P$  applied to matrix  $A$  results in

$$PA = \left[ \begin{array}{c|c} B & C \\ \hline D & E \end{array} \right],$$

where the rectangular matrix  $B \in \text{Mat}(r, b, \mathbb{F}_2)$ , with  $r \leq b$ , has full rank and it satisfies  $D = YB$  for an opportune matrix  $Y$  which we have to determine.

In case  $r = b$ , since  $B \in \text{GL}(b, \mathbb{F}_2)$  we easily deduce that  $Y = DB^{-1}$ . Instead, when the matrix  $B$  is full rank with less rows than columns, a *pseudo-inverse*  $B^\dagger$  has to be computed. For example the pseudo-inverse of Moore-Penrose [Ben-Israel and Greville 2003] is given by

$$B^\dagger = B^T(BB^T)^{-1}$$

and satisfies

$$\begin{aligned} BB^\dagger &= BB^T(BB^T)^{-1} = I, \\ DB^\dagger &= YBB^\dagger = Y. \end{aligned}$$

Thus,  $Y$  can be computed by simple right multiplication by a pseudo-inverse of  $B$ . Although the use of the Moore-Penrose's pseudo-inverse is correct, a more efficient pseudo-inverse can be constructed. Let  $J \in \text{Mat}(b, r, \mathbb{F}_2)$  be an *insertion* matrix whose effect is to insert  $b - r$  zero-rows into a matrix with  $r$  rows. Let  $R \in \text{Mat}(b, b, \mathbb{F}_2)$  be the matrix containing linearly independent rows which makes the square matrix  $JB + R$  non-singular. Notice that  $J^T$  is a *projection* matrix which satisfies

$$J^T J = I, \quad J^T R = 0. \quad (6)$$

If  $r = b$ , i.e. if  $B$  is square and non-singular, insertions are not necessary and we have  $J = I$ ,  $R = 0$ .

EXAMPLE 3.3. In case  $b = 3$  and  $r = 2$  we can see a situation as

$$J_j = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad R_j = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Let consider  $Z = (JB + R)^{-1}J$ . Due to the properties in (6) and the relation  $D = YB$ , we have

$$\begin{aligned} BZ &= B(JB + R)^{-1}J = J^T JB(JB + R)^{-1}J \\ &= J^T (JB + R)(JB + R)^{-1}J = J^T J = I \\ DZ &= YBZ = Y. \end{aligned}$$

Thus the matrix  $Z$  has the same effect of the pseudo-inverse (it is a pseudo-inverse different from the Moore-Penrose one) and it is used in the computation of  $Y$  during the factorization procedure. In order to build  $Z$ , we need an algorithm to compute the inverse of a (small) square matrix in  $\mathbb{F}_2$ ; it will be treated in the next section.

### 3.2 Incremental construction of the inverse of a square matrix in $\mathbb{F}_2$

Let  $B \in \text{GL}(b, \mathbb{F}_2)$  with all principal minors not-singular, it is possible to incrementally build its inverse  $Z$ . This requirement is not restrictive because every non-singular matrix by a row permutation satisfies it (due to Gauss *LU* decomposition,



see [Kincaid and Cheney 2002] page 156 Theorem 1). Let  $B_k$  be the  $k^{\text{th}}$  principal minor of  $B$  and  $Z_k$  its inverse. We can directly obtain  $B_{k+1}^{-1}$  from  $B_k^{-1}$  using the following factorization

$$B_{k+1} = \begin{bmatrix} B_k & c \\ r^T & \alpha \end{bmatrix} = \begin{bmatrix} I & 0 \\ r^T Z_k & 1 \end{bmatrix} \begin{bmatrix} B_k & c \\ 0 & 1 \end{bmatrix} \quad (7)$$

which holds when

$$\alpha \oplus r^T Z_k c = 1. \quad (8)$$

*Remark 3.4.* In case of  $\mathbb{F}_q$ , the previous condition becomes  $\alpha - r^T Z_k c = \beta$  where  $\alpha, \beta \in \mathbb{F}_q$  and  $\beta \neq 0$ . Moreover, the last block in (7) becomes

$$\begin{bmatrix} B_k & c \\ 0 & \beta \end{bmatrix}.$$

Inverting factorization (7) we get immediately:

$$B_{k+1}^{-1} = \begin{bmatrix} B_k & c \\ r^T & \alpha \end{bmatrix}^{-1} = \begin{bmatrix} Z_k & Z_k c \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ r^T Z_k & 1 \end{bmatrix} \quad (9)$$

(Notice that  $B_1$  is a  $1 \times 1$  matrix and thus  $B_1 = Z_1 = [1]$ ). Therefore, due to (9) we can compute  $Z_{k+1}$  from  $Z_k$  as

$$Z_{k+1} = B_{k+1}^{-1} = \begin{bmatrix} Z_k \oplus (Z_k c)(r^T Z_k) & Z_k c \\ r^T Z_k & 1 \end{bmatrix}, \quad (10)$$

which needs two matrix-vector multiplication and a rank one update. Moreover, setting  $\tilde{c} = Z_k c$ , the last matrix in (10) can be written as a matrix-matrix product:

$$Z_{k+1} = \begin{bmatrix} Z_k \oplus \tilde{c}(r^T Z_k) & \tilde{c} \\ r^T Z_k & 1 \end{bmatrix} = H_{k+1} \begin{bmatrix} Z_k & 0 \\ 0 & 1 \end{bmatrix}, \quad H_{k+1} = \begin{bmatrix} I \oplus \tilde{c} r^T & \tilde{c} \\ r^T & 1 \end{bmatrix}. \quad (11)$$

Let  $M_k$  be the matrix obtained multiplying by  $Z_k$  the first  $k$  rows of  $B$ :

$$M_k = \begin{bmatrix} I & Z_k c & Z_k C \\ r^T & \alpha & e^T \\ D & d & E \end{bmatrix} = \begin{bmatrix} I & \tilde{c} & \tilde{C} \\ r^T & \alpha & e^T \\ D & d & E \end{bmatrix}, \quad \text{where } B = \begin{bmatrix} B_k & c & C \\ r^T & \alpha & e^T \\ D & d & E \end{bmatrix}.$$

Due to (11), the update of  $M_{k+1}$  results in

$$M_{k+1} = \begin{bmatrix} Z_{k+1} & 0 \\ 0 & I \end{bmatrix} B = \begin{bmatrix} H_{k+1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} Z_k & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & I \end{bmatrix} B = \begin{bmatrix} H_{k+1} & 0 \\ 0 & I \end{bmatrix} M_k$$

and thus,  $\mathbf{H}_{k+1}$  is used in updating both  $\mathbf{M}_{k+1}$  and  $\mathbf{Z}_{k+1}$ . Moreover, we obtain the following update formulas:

$$\mathbf{H}_{k+1} \begin{bmatrix} \tilde{\mathbf{C}} \\ \mathbf{e}^T \end{bmatrix} = \begin{bmatrix} \mathbf{I} \oplus \tilde{\mathbf{c}}\mathbf{r}^T & \tilde{\mathbf{c}} \\ \mathbf{r}^T & 1 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{C}} \\ \mathbf{e}^T \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{C}} \oplus \tilde{\mathbf{c}}\mathbf{s}^T \\ \mathbf{s}^T \end{bmatrix}, \quad \text{where } \mathbf{s}^T = \mathbf{r}^T \tilde{\mathbf{C}} \oplus \mathbf{e}^T.$$

Note that it is possible to perform the incremental build when all principal minors are non-singular, which is equivalent to satisfy equation (8) for all  $k$ . Such a condition is used to dynamically select linear independent rows in the construction of  $\mathbf{B}^{(j)}$ .

### 3.3 Construction of row permutation

The steps of reduction from  $\mathbf{A}^{(j)}$  to  $\mathbf{A}^{(j+1)}$  need the computation of permutation matrix  $\mathbf{P}^{(j)}$  which permutes the rows of  $\mathbf{A}^{(j)}$  in order to obtain  $\mathbf{P}^{(j)}\mathbf{A}^{(j)}$  partitioned as in (3). The block  $\mathbf{B}^{(j)}$  must be non-singular with all principal minors non-singular. This permutation can be computed using only the first  $b$  columns of  $\mathbf{A}^{(j)}$  and must satisfy

$$\mathbf{P}^{(j)}\mathbf{A}_{\bullet,0..b}^{(j)} = \begin{bmatrix} \mathbf{B}^{(j)} \\ \mathbf{D}^{(j)} \end{bmatrix}.$$

The selection of the rows and the construction of the inverse of  $\mathbf{B}^{(j)}$  are done together using incremental update of Section 3.2, where the selected rows must satisfy condition (8). Observe that to check condition (8), the  $\ell$ -row of the sub-matrix  $\mathbf{A}_{\bullet,0..b}^{(j)}$  is partitioned as  $\mathbf{A}_{\ell,0..b}^{(j)} = (\mathbf{r}^T, \alpha, \beta^T)$  where the  $\beta^T$  portion of the row is ignored in the computation.

### 3.4 Insertion of linearly independent rows

In the computation of row permutation it may happen that condition (8) is not satisfied by all the rows of the column block  $\mathbf{A}_{\bullet,0..b}^{(j)}$ . Obviously, such a situation does not arise if  $n \geq m$  and when the matrix is full rank.

At this stage, standard algorithms introduce column permutations to satisfy condition (8). If such a condition cannot be satisfied even using column permutations, it means that last rows are linearly dependent on the previous ones and then the algorithm must end. Column permutations are not executed in our algorithm, so that new linearly independent rows are inserted using the two matrices  $\mathbf{J}$  and  $\mathbf{R}$ , introduced in Section 3.1. We observe that it is easy to build a row that satisfies condition (8) and that, in particular, the row  $(\mathbf{r}^T, \alpha, \beta^T) = (\mathbf{0}^T, 1, \mathbf{0}^T)$  trivially satisfies it. In practice, the presented process mixes row permutations and row insertions; it can be respectively split in a row permutations followed by a rows insertions:

$$\mathbf{P}^{(j)}\mathbf{A}_{\bullet,0..b}^{(j)} = \begin{bmatrix} \mathbf{B}^{(j)} \\ \mathbf{D}^{(j)} \end{bmatrix}, \quad \mathbf{J}_j\mathbf{B}^{(j)} + \mathbf{R}_j.$$

Notice that the permutation  $\mathbf{P}^{(j)}$  and the insertion  $\mathbf{J}_j$  are chosen in such a way the square block  $\mathbf{J}_j\mathbf{B}^{(j)} + \mathbf{R}_j$  has all principal minors non-singular and satisfies (6).

The following procedure performs the operation described in Sections 3.2-3.3 and 3.4 to obtain an incremental construction of the pseudo-inverse  $Z$ .

| Procedure buildZ( $\mathcal{A}, r$ ) |  |
|--------------------------------------|--|
| 1                                    | $i_b \leftarrow 0;$  |
| 2                                    | <b>for</b> $j = 0..b$ <b>do</b> $\mathcal{Z}_j \leftarrow 2^j; \mathcal{M}_j \leftarrow 2^j; \mathcal{P}_j \leftarrow -1;$   |
| 3                                    | <b>for</b> $k_{\text{bit}} = 0..b$ <b>do</b>   |
| 4                                    | $c \leftarrow 2^{k_{\text{bit}}};$   |
| 5                                    | <b>for</b> $i = 0..k_{\text{bit}}$ <b>do</b>   |
| 6                                    | <b>if</b> $\mathcal{M}_i \odot 2^{k_{\text{bit}}} \neq 0$ <b>then</b> $c \leftarrow c \oplus 2^i;$   |
| 7                                    | <b>end</b>   |
| 8                                    | <b>for</b> $i = i_b..r$ <b>do</b>  |
| 9                                    | <b>if</b> popCount( $c \odot \mathcal{A}_i$ ) is odd <b>then</b>   |
| 10                                   | $\mathcal{P}_{k_{\text{bit}}} \leftarrow i; \mathcal{A}_i \rightleftharpoons \mathcal{A}_{i_b}; \mathcal{M}_{k_{\text{bit}}} \leftarrow \mathcal{A}_{i_b}; i_b \leftarrow i_b + 1; \text{break};$                                  |
| 11                                   | <b>end</b>   |
| 12                                   | <b>end</b>   |
| 13                                   | $y \leftarrow \mathcal{M}_{k_{\text{bit}}};$ // Update of $\mathcal{Z}$ and $\mathcal{M}$  |
| 14                                   | <b>for</b> $j = 0..k_{\text{bit}}$ <b>do</b>   |
| 15                                   | <b>if</b> $y \odot 2^j \neq 0$ <b>then</b> $\mathcal{Z}_{k_{\text{bit}}} \leftarrow \mathcal{Z}_{k_{\text{bit}}} \oplus \mathcal{Z}_j; \mathcal{M}_{k_{\text{bit}}} \leftarrow \mathcal{M}_{k_{\text{bit}}} \oplus \mathcal{M}_j;$ |
| 16                                   | <b>end</b>   |
| 17                                   | <b>for</b> $j = 0..k_{\text{bit}}$ <b>do</b>   |
| 18                                   | <b>if</b> $c \odot 2^j \neq 0$ <b>then</b> $\mathcal{Z}_j \leftarrow \mathcal{Z}_{k_{\text{bit}}} \oplus \mathcal{Z}_j; \mathcal{M}_j \leftarrow \mathcal{M}_{k_{\text{bit}}} \oplus \mathcal{M}_j;$                               |
| 19                                   | <b>end</b>   |
| 20                                   | <b>end</b>   |
| 21                                   | $m \leftarrow 0;$  |
| 22                                   | <b>for</b> $i = 0..b$ <b>do</b>  |
| 23                                   | <b>if</b> $\mathcal{P}_i = -1$ <b>then</b>   |
|                                      | // $\overline{m}$ is the integer with bits complemented  |
| 24                                   | <b>for</b> $j = 0..b$ <b>do</b> $\mathcal{Z}_j \leftarrow (\mathcal{Z}_j / 2 \odot \overline{m}) \oplus (\mathcal{Z}_j \odot m);$  |
| 25                                   | <b>else</b>  |
| 26                                   | $m \leftarrow 2m + 1;$   |
| 27                                   | <b>end</b>   |
| 28                                   | <b>end</b>   |

#### 4. BLOCK RECURSIVE ALGORITHM

In this section we describe a recursive version of the algorithm presented in Section 3. Let consider  $\mathbf{A} \in \text{Mat}(n, m, \mathbb{F}_2)$  splitted into two sub-matrices  $\mathbf{A}_L \in \text{Mat}(n, p, \mathbb{F}_2)$  and  $\mathbf{A}_R \in \text{Mat}(n, m - p, \mathbb{F}_2)$  with  $p = \lceil m/2 \rceil$  such that

$$\mathbf{A} = [\mathbf{A}_L | \mathbf{A}_R].$$

Applying the decomposition in Lemma 4.1 to the left part  $\mathbf{A}_L$ , we have  $\mathbf{P}\mathbf{A}_L = \mathbf{L}\mathbf{U}$  and so

$$\mathbf{P}\mathbf{A} = [\mathbf{P}\mathbf{A}_L | \mathbf{P}\mathbf{A}_R] = [\mathbf{L}\mathbf{U} | \mathbf{P}\mathbf{A}_R].$$

Since  $PA_R = \begin{bmatrix} C \\ D \end{bmatrix}$ , where  $C \in \text{Mat}(r, m-p, \mathbb{F}_2)$  and  $D \in \text{Mat}(n-r, m-p, \mathbb{F}_2)$ , we obtain

$$PA = \left[ \begin{pmatrix} L_{0..r, \bullet} \\ L_{r..n, \bullet} \end{pmatrix} U \middle| \begin{pmatrix} C \\ D \end{pmatrix} \right] = \left[ \begin{array}{c|c} L_{0..r, \bullet} & 0 \\ \hline L_{r..n, \bullet} & I \end{array} \right] \left[ \begin{array}{c|c} U & L_{0..r, \bullet}^{-1} C \\ \hline 0 & D \oplus L_{r..n, \bullet} L_{0..r, \bullet}^{-1} C \end{array} \right].$$

Let  $A' = D \oplus L_{r..n, \bullet} L_{0..r, \bullet}^{-1} C$ , we can recursively apply the factorization  $P'A' = L'U'$  and we obtain

$$PA = \left[ \begin{array}{c|c} L_{0..r, \bullet} & 0 \\ \hline L_{r..n, \bullet} & I \end{array} \right] \left[ \begin{array}{c|c} U & L_{0..r, \bullet}^{-1} C \\ \hline 0 & A' \end{array} \right] = \left[ \begin{array}{c|c} L_{0..r, \bullet} & 0 \\ \hline L_{r..n, \bullet} & (P')^{-1} \end{array} \right] \left[ \begin{array}{c|c} U & L_{0..r, \bullet}^{-1} C \\ \hline 0 & L'U' \end{array} \right]$$

and then

$$\left[ \begin{array}{c|c} I & 0 \\ \hline 0 & P' \end{array} \right] PA = \left[ \begin{array}{c|c} L_{0..r, \bullet} & 0 \\ \hline P' L_{r..n, \bullet} & L' \end{array} \right] \left[ \begin{array}{c|c} U & L_{0..r, \bullet}^{-1} C \\ \hline 0 & U' \end{array} \right].$$

*Remark 4.1.* Notice that here we perform the matrix-matrix multiplications using our implementation of the Strassen's algorithm.

## 5. PERFORMANCE TUNING

The algorithm presented in Section 3 completely avoids column permutations and the cost of the decomposition is given by the cost of the matrix-matrix multiplication. The cost to perform the M4RM algorithm to multiply  $A \in \text{Mat}(n, b, \mathbb{F}_2)$  and  $B \in \text{Mat}(b, b, \mathbb{F}_2)$  is approximately given by the costs of the Xor operations. In particular, if we neglect the cost of the memory access and other minor costs, we obtain that

$$\text{cost of M4RM} = T_c^b + nR_c^b, \quad T_c^b = \left(\frac{b}{c}\right) (2^c - 1) \text{Xor}_b, \quad R_c^b = \left(\frac{b}{c}\right) \text{Xor}_b \quad (12)$$

where

- (1)  $c$  is the size of the tables used to perform M4RM algorithm;
- (2)  $T_c^b$  is the cost of the tables construction;
- (3)  $R_c^b$  is the cost of the rows operations;
- (4)  $\text{Xor}_b$  is the cost of the Xor operation using integer of  $b$  bit size.

In the following subsection we are going to discuss about the choices of the parameters  $b$  and  $c$ . Moreover, we will give an idea about the switching point from the M4RM multiplication and the Strassen multiplication.

### 5.1 Integer word-size and performance of M4RM

Let us consider the two matrices  $\mathbf{A} \in \text{Mat}(n, 2b, \mathbb{F}_2)$  and  $\mathbf{B} \in \text{Mat}(2b, 2b, \mathbb{F}_2)$ . Due to (12) and using  $2b$  bits as word-size, the cost to perform the M4RM to multiply  $\mathbf{A}$  and  $\mathbf{B}$  is approximately  $T_c^{2b} + nR_c^{2b}$ . On the other hand, using a word-size of  $b$  bits, the cost of M4RM becomes  $4T_c^b + n4R_c^b$ .

Considering the cost of the rows contribution and the respective ratio, we have that

$$\frac{\text{Cost of } b\text{-bits row}}{\text{Cost of } 2b\text{-bits row}} = \frac{4nR_c^b}{nR_c^{2b}} = \frac{2\text{Xor}_b}{\text{Xor}_{2b}}$$

and so, in case of  $\text{Xor}_{2b} < 2\text{Xor}_b$ , it is convenient to use  $2b$  bits.

The cost of the construction of the tables is given by  $4T_c^b$  and  $T_c^{2b}$  and the corresponding ratio is then

$$\frac{\text{Cost per } b\text{-bits tables}}{\text{Cost per } 2b\text{-bits tables}} = \frac{4T_c^b}{T_c^{2b}} = \frac{2\text{Xor}_b}{\text{Xor}_{2b}}.$$

Since the cost of the  $\text{Xor}_b \approx \text{Xor}_{2b}$ , it is convenient to chose  $2b$  as word-size.

Once the size  $b$  has been chosen, should be more convenient to choose the size  $c$  of the tables that are used for the M4RM algorithm. Since the cost to perform the product of the two matrices  $\mathbf{A} \in \text{Mat}(n, b, \mathbb{F}_2)$  and  $\mathbf{B} \in \text{Mat}(b, b, \mathbb{F}_2)$  is given by (12), the optimal table size  $c$  (when  $b$  and  $n$  are known) can be estimated minimizing (12), which is the minimum of the following function

$$C(b, c, n) = \left(\frac{b}{c}\right) (2^c - 1 + n).$$

*Remark 5.1.* Actually, to determine the minimum of  $C$  is complicated. However, it can be observed that  $C$ , as a function of  $n$ , is a straight line with a slope that decreases when  $c$  increases. So, when  $n$  exceeds the value  $C(b, c, n) = C(b, c+1, n)$ , it is convenient to use a table of size  $c+1$  instead of size  $c$ . In the case of the product of square matrices, the minimum cost is obtained minimizing  $C(b, c, b)$  (with respect to  $c$ ) and we obtain the following values:

$$\arg \min_c \{C(32, c, 32)\} \approx 4.08 \quad \arg \min_c \{C(64, c, 64)\} \approx 4.77$$

$$\arg \min_c \{C(128, c, 128)\} \approx 5.5.$$

In Table II we report the costs to perform product of square matrices of size 32, 64 and 128.

### 5.2 How to choose switching point for Strassen Matrix-Matrix multiplication

The cost to multiply  $\mathbf{A} \in \text{Mat}(n, b, \mathbb{F}_2)$  and  $\mathbf{B} \in \text{Mat}(b, b, \mathbb{F}_2)$  using the M4RM algorithm is expressed in (12). Starting by the previous formula, we can easily obtain the cost to multiply (with M4RM algorithm) two matrices of size  $n$ .

Let  $N = n/b$  be the number of blocks in which we divide the matrix. We must apply  $N^2$  times the M4RM algorithm and we obtain the following cost

$$M(n) = N^2(T_c^b + nR_c^b) = \frac{\text{Xor}_b}{b c} (n^2(2^c - 1) + n^3).$$

Table I. Cost of tables construction and cost of the rows operation for M4RM.

|    | 32 bits |                   | 64 bits |                   | 128 bits |                   |
|----|---------|-------------------|---------|-------------------|----------|-------------------|
|    | $T_c$   | $1000 \times R_c$ | $T_c$   | $1000 \times R_c$ | $T_c$    | $1000 \times R_c$ |
| 2  | 0.028   | 10.22             | 0.048   | 15.80             | 0.086    | 49.56             |
| 3  | 0.044   | 6.25              | 0.066   | 10.77             | 0.151    | 34.19             |
| 4  | 0.069   | 5.18              | 0.099   | 9.71              | 0.223    | 21.89             |
| 5  | 0.140   | 3.78              | 0.151   | 7.30              | 0.349    | 19.89             |
| 6  | 0.222   | 2.91              | 0.254   | 6.07              | 0.688    | 17.25             |
| 7  | 0.252   | 2.82              | 0.465   | 5.06              | 2.322    | 15.72             |
| 8  | 0.484   | 2.44              | 0.721   | 4.73              | 3.734    | 14.52             |
| 9  | 0.772   | 2.40              | 1.709   | 4.37              | 7.446    | 16.12             |
| 10 | 4.227   | 1.83              | 3.465   | 4.98              | 11.364   | 16.87             |

| Normalized cost |                 |                    |                |                   |        |                   |
|-----------------|-----------------|--------------------|----------------|-------------------|--------|-------------------|
|                 | $16 \times T_c$ | $16000 \times R_c$ | $4 \times T_c$ | $4000 \times R_c$ | $T_c$  | $1000 \times R_c$ |
| 2               | 0.444           | 164.80             | 0.192          | 62.90             | 0.086  | 49.40             |
| 3               | 0.700           | 101.20             | 0.264          | 43.10             | 0.151  | 34.06             |
| 4               | 1.112           | 83.00              | 0.398          | 38.82             | 0.223  | 23.08             |
| 5               | 2.248           | 60.36              | 0.606          | 29.20             | 0.349  | 19.89             |
| 6               | 3.560           | 45.80              | 1.016          | 24.26             | 0.688  | 17.28             |
| 7               | 4.028           | 46.64              | 1.862          | 20.30             | 2.322  | 15.73             |
| 8               | 7.748           | 38.20              | 2.886          | 19.20             | 3.734  | 14.49             |
| 9               | 12.348          | 37.64              | 6.838          | 17.20             | 7.446  | 16.16             |
| 10              | 67.640          | 29.32              | 13.862         | 20.04             | 11.364 | 16.83             |

Time measured in microseconds

To perform Strassen matrix-matrix multiplication algorithm, our implementation needs essentially 22 additions and 7 multiplications of matrices having size  $\frac{n}{2}$ . Since the cost to compute one addition is given by  $nNX_{\text{or}_b}/4$ , the whole cost for Strassen's algorithm is

$$S(n) = \frac{11n^2}{2b}X_{\text{or}_b} + 7S\left(\frac{n}{2}\right).$$

Then, it is convenient to use Strassen's algorithm as long as  $S(n) \leq M(n)$  or

$$S(n) = \frac{11n^2}{2b}X_{\text{or}_b} + 7S(n/2) \leq \frac{11n^2}{2b}X_{\text{or}_b} + 7M(n/2) \leq M(n).$$

In other words, it is convenient to use Strassen's algorithm when

$$n \geq 44c + 6(2^c - 1).$$

Table II. Cost of the product of 2 square matrices having size respectively 32, 64, 128. The normalized costs are also reported.

| bit | 32 bit  |                  | 64 bit  |                 | 128 bit   |
|-----|---------|------------------|---------|-----------------|-----------|
|     | 32 × 32 | normalized (×64) | 64 × 64 | normalized (×8) | 128 × 128 |
| 2   | 0.362   | 22.88            | 1.060   | 8.48            | 6.43      |
| 3   | 0.255   | 16.16            | 0.755   | 6.04            | 4.51      |
| 4   | 0.240   | 15.20 *          | 0.725   | 5.80            | 2.88      |
| 5   | 0.262   | 16.64            | 0.625   | 5.00 *          | 2.90      |
| 6   | 0.312   | 20.00            | 0.645   | 5.12            | 2.89 *    |
| 7   | 0.345   | 22.08            | 0.790   | 6.32            | 3.91      |
| 8   | 0.570   | 36.64            | 1.025   | 8.16            | 5.52      |
| 9   | 0.845   | 54.40            | 1.955   | 15.60           | 9.57      |
| 10  | 4.355   | 279.80           | 3.635   | 29.08           | 13.56     |

## 6. PERFORMANCE TESTS AND COMPARISON

To evaluate the performance of our algorithm, we compute the decomposition of  $n \times n$  matrices with entries in  $\mathbb{F}_2$ . Our block decomposition works well both for dense matrices and for relatively sparse matrices. Notice that in the former case the obtained matrices have rank most probably equal to  $n - 1$  (or  $n$ ); in the latter case we have low-rank matrices.

First, we have constructed a sample of random dense matrices of size  $n$  (where the size  $n$  ranges from 256 and 65536).

In Table III we give the minimum value of ten observed running times for computing our block decomposition (recursive and non-recursive), in case  $b = 32, 64, 128$ .

Table III. Compare running times with  $b = 32, 64, 128$  recursive (rec) and non recursive versions of our algorithm. Execution obtained using LLVM 3.0 compiler (MAC OSX) on a 3.06GHz Intel(R) Xeon.

|          | 32           | 32 rec  | 64      | 64 rec  | 128     | 128 rec |
|----------|--------------|---------|---------|---------|---------|---------|
| <i>n</i> | milliseconds |         |         |         |         |         |
| 256      | 0.106        | 0.108   | 0.131   | 0.133   | 0.247   | 0.250   |
| 512      | 0.347        | 0.350   | 0.327   | 0.330   | 0.553   | 0.557   |
| 1024     | 1.574        | 1.741   | 1.070   | 1.171   | 1.443   | 1.527   |
| 2048     | 9.260        | 10.862  | 5.076   | 5.927   | 5.172   | 5.795   |
| 4096     | 64.849       | 76.458  | 32.454  | 37.996  | 27.274  | 30.792  |
| <i>n</i> | seconds      |         |         |         |         |         |
| 8192     | 0.492        | 0.549   | 0.248   | 0.267   | 0.203   | 0.206   |
| 16384    | 3.922        | 3.981   | 1.939   | 1.981   | 1.646   | 1.523   |
| 32768    | 32.011       | 29.791  | 23.859  | 15.440  | 12.884  | 11.498  |
| 65536    | 253.169      | 216.323 | 190.065 | 114.794 | 102.071 | 86.156  |

In Table IV, the minimum running time of ten trials to obtain a matrix decompo-

sition is given. In particular, we compare the running times to obtain M4RI, PLUQ, PLE decompositions adopted into SAGE [Stein et al. 2012] with the ones to get our 128 block recursive decomposition.

Table IV. Compare running times with three algorithms (PLUQ, M4RI, PLE) adopted into SAGE and our recursive algorithm with  $b = 128$  using LLVM 3.0 compiler (MAC OSX) on a 3.06GHz Intel(R) Xeon.

|        | M4RI         | PLUQ    | PLE     | our    |
|--------|--------------|---------|---------|--------|
| $n$    | milliseconds |         |         |        |
| 256    | 0.248        | 0.239   | 0.239   | 0.252  |
| 512    | 0.847        | 0.836   | 0.821   | 0.562  |
| 1 024  | 3.428        | 2.653   | 2.551   | 1.539  |
| 2 048  | 12.810       | 10.509  | 10.003  | 5.850  |
| 4 096  | 62.231       | 53.686  | 50.400  | 31.406 |
| $n$    | seconds      |         |         |        |
| 8 192  | 0.349        | 0.340   | 0.334   | 0.207  |
| 16 384 | 2.462        | 2.425   | 2.392   | 1.532  |
| 32 768 | 18.674       | 18.558  | 18.341  | 11.529 |
| 65 536 | 135.906      | 128.567 | 125.783 | 86.538 |

Then, we constructed low-rank matrices as follows. We consider the samples

$$S_n = \{S_{n,i} \mid i = 2, \dots, n\}$$

of 39 relatively sparse matrices of size  $n$  having respectively  $i$  non-zero elements per rows.

In Figure 2 we have plotted the observed running times (in milliseconds) for M4RI (cross), PLUQ (square), PLE(circles) and our 128 bit block recursive (triangles) decomposition. This is done for every matrix in the sample  $S_n$ . The size  $n$  ranges from 1024 to 65536.

#### Acknowledgment

The authors would like to thank the following people for their valuable comments and suggestions: Prof. M. Elia, Prof. M. Sala and Prof. T. Muntean.

Part of the second author's research has been funded by the "Groupe d'Études et Recherche en Informatique des Systèmes Communicants Sécurisés", Aix-Marseille Université.

#### REFERENCES

- AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. 1974. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Publishing Company, Boston, Massachusett.
- ALBRECHT, M., BARD, G., AND HART, W. 2010. Algorithm 898: Efficient multiplication of dense matrices over  $\text{GF}(2)$ . *ACM Trans. Math. Softw.* 37, 9:1–9:14.
- ALBRECHT, M. R., BARD, G. V., AND PERNET, C. 2011. Efficient dense gaussian elimination over the finite field with two elements. Tech. rep., Cornell University Library.
- ARLAZAROV, V. L., DINIC, E. A., KRONROD, M. A., AND FARADŽEV, I. A. 1970. On economical construction of the transitive closure of a directed graph. *Soviet Mathematics—Doklady* 11, 5, 1209–1210.
- BACH, E. AND SHALLIT, J. 1996. *Algorithmic Number Theory: Efficient algorithms*. Foundations of computing. MIT Press, Cambridge, USA.



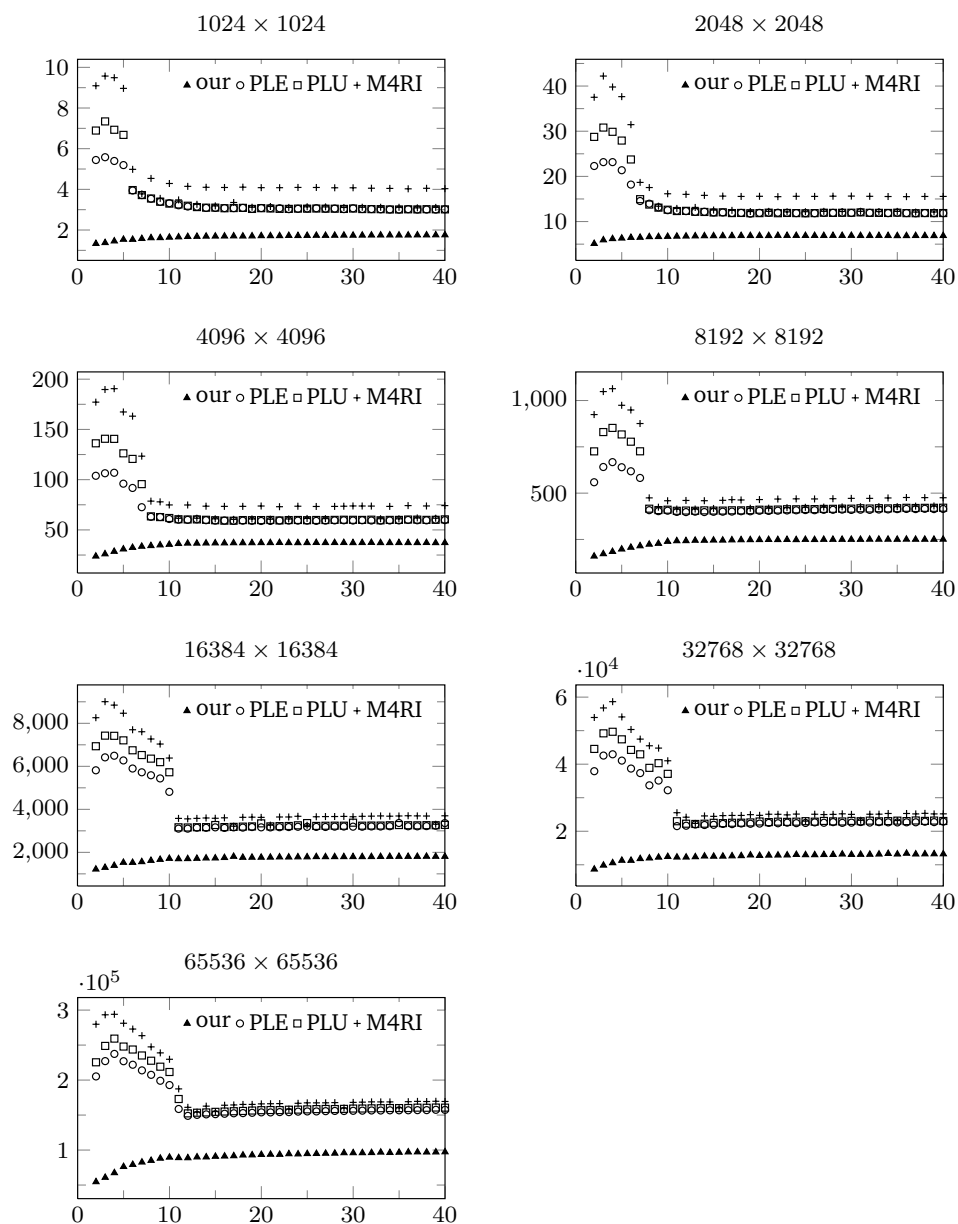


Fig. 2. Running times for computing a matrix decomposition of  $n \times n$  low-rank matrices using LIVM 3.0 compiler (MAC OSX) on a 2.53GHz Intel(R) Core 2 Duo.

BEN-ISRAEL, A. AND GREVILLE, T. N. 2003. *Generalized inverses. Theory and applications. 2nd ed.* CMS Books in Mathematics. Springer.

COPPERSMITH, D. AND WINOGRAD, S. 1990. Matrix multiplication via arithmetic progressions. *J. Symbolic Comput.* 9, 3, 251–280.

- DOUGLAS, C. C., HEROUX, M., SLISHMAN, G., AND SMITH, R. M. 1994. GEMMW: a portable level 3 blas winograd variant of strassen's matrix-matrix multiply algorithm. *Journal of Computational Physics* 110, 1, 1 – 10.
- GOLUB, G. H. AND VAN LOAN, C. F. 1996. *Matrix computations*, Third ed. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD.
- HAYNSWORTH, E. 1968. *On The Schur Complement*. Defense Technical Information Center, Basel University.
- HIGHAM, N. J. 1990. Exploiting fast matrix multiplication within the level 3 blas. *ACM Trans. Math. Softw.* 16, 352–368.
- HUSS-LEDERMAN, S., JACOBSON, E. M., TSAO, A., TURNBULL, T., AND JOHNSON, J. R. 1996. Implementation of strassen's algorithm for matrix multiplication. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*. Supercomputing '96. IEEE Computer Society, Washington, DC, USA.
- JEANNEROD, C.-P., PERNET, C., AND STORJOHANN, A. 2011. Rank-profile revealing gaussian elimination and the cup matrix decomposition. Tech. rep., Cornell University Library.
- KAPORIN, I. 1999. A practical algorithm for faster matrix multiplication. *Numerical Linear Algebra with Applications* 6, 8, 687–700.
- KINCAID, D. AND CHENEY, E. 2002. *Numerical Analysis: Mathematics of Scientific Computing*. Pure and applied undergraduate texts. American Mathematical Society.
- MEYER, C. D. 2000. *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.
- SHOUP, V. 2009. *A computational introduction to number theory and algebra*. Cambridge University Press.
- STEIN, W. ET AL. 2012. *Sage Mathematics Software (Version 5.2)*. The Sage Development Team. <http://www.sagemath.org>.
- STRASSEN, V. 1969. Gaussian elimination is not optimal. *Numerische Mathematik* 13, 354–356. 10.1007/BF02165411.
- WINOGRAD, S. 1971. On multiplication of  $2 \times 2$  matrices. *Linear Algebra and its Applications* 4, 4, 381 – 388.
- ZHANG, F. 2005. *The Schur Complement and Its Applications*. Numerical Methods and Algorithms. Springer Science.